

General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.
- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.
- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.
- This document is paginated as submitted by the original source.
- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

(NASA-CR-173207) ERROR LATENCY ESTIMATION
USING FUNCTIONAL FAULT MODELING (Iowa Univ.)
27 p HC A03/HF A01 CSCL 12A

N84-16856

Unclas
G3/64 00534

Error Latency Estimation Using
Functional Fault Modeling

By

Sridhar R. Manthani
Nirmal R. Saxena
John P. Robinson

Electrical and Computer Engineering
The University of Iowa
Iowa City, Iowa 52242



Prepared for
Langley Research Center
Under NAG-195

August 1983

Summary

A complete modeling of faults at gate level for a fault-tolerant computer is both infeasible and uneconomical. Functional Fault modeling is an approach where units are characterized at an intermediate level and then combined to determine fault behavior. This report is a preliminary study on the applicability of Functional Fault Modeling to the FTMP. Using this model a forecast of error latency is made for some functional blocks. This approach may be useful in representing larger sections of the hardware and may aid in uncovering system-level deficiencies.

Introduction

The complexity of a fault-tolerant computer makes it impractical to exhaustively estimate its reliability parameters using a low level fault model (1,2). A particular logic function may be implemented in different forms as a technology matures or separate vendors may select different yet equivalent forms. Even in current MSI circuits, equivalent logic specifications are used rather than detailed transistor representations. A higher level approach is Functional Fault Modeling (3-5). In a Functional Model gates cease to be the primitives in analysis. The whole hardware structure is partitioned into functional primitives. The partitioning is based on many factors. Whatever partitioning was used in the design process is usually the starting point for functional fault modeling. Faults occurring internal to a partition propagate to the outputs of the partition (all gate level faults will manifest as some functional level faults). This mapping of faults is many to one thus making functional level modeling potentially simpler.

If the technology is well understood then the functional model can be made to accurately represent the partition both in range of behavior and in statistical characteristics. A library of models can be developed including a hierarchy of units (e.g., flip-flop models used to develop a counter model, etc.). When the technology is not well understood the functional modeling can be done in a more conservative manner (6-11). All conceivable fault behavior could be represented, then as actual behavior data becomes available the functional model can be made more accurate.

Because of the highly redundant nature of the FTMP, it is difficult to surface many of its faults during its normal operation. But FTMP behaves as a fault secure circuit for most of the faults. Eventually when that redundant

portion of the circuit is exercised the fault affect propagates to the outputs of the partition. Since we know the circuit structure it is easy to break an existing functional primitive into smaller primitives and improve the accuracy of the model.

FACTORS INFLUENCING ERROR LATENCY

In FTMP, Error Latency is influenced by hardware and software. Faults on an active single bus line are masked by the voters and except in the BGU's, the error decoding circuits detect these errors.

Three primary factors influence error latency:

1. Whether the Faulty circuit is exercised,
2. The rate at which the error latches are read, and
3. Whether that part is influencing the detection circuitry.

Each of the above cases needs to be analysed carefully to arrive at an accurate overall latency estimate.

1. The first factor is a natural consequence of the highly redundant nature of the FTMP. If the fault occurs in an inactive region it is harmless, but another fault at this juncture might cause abnormal behaviour. An example of this is a fault on one of the spare buses as the first fault followed by a fault on an active bus. This might result in replacing all the units enabled on the faulty active bus by another faulty bus. Subsequent detection of the second fault might take several cycles. It is imperative to activate periodically all the redundant parts just to avoid long latency faults. Some assumptions were made regarding exercising the redundant parts to arrive at a definitive figures for the error latency. The particular assumptions are given later.

2. The second factor is software determined. A program called SCC (System Configuration Controller) performs the chore of reading and interpreting the error latches. The dispatcher directs SCC to the leastly loaded triad. The first part of the program determines whether any errors were reported in the preceding frame. It also determines whether the error reported during a previous frame has been corrected. If not it waits for a maximum of four cycles for the previous error to be corrected. An assumption is made in this regard to make definitive forecasts.

3. Some parts like the BGUs do not have error detection circuitry. Single faults here are often masked although all parts may be active. These latent faults have to be dealt with specially. Normally there is no way to propagate many of the single faults in the voting hardware and deskewers unless we have another fault in such a way that they cooperate to cause noticeable faulty output behaviour.

DESCRIPTION OF DETECTION PROCESS AT HIGH LEVEL

It may be recalled that all tasks run at one of the three preassigned rates. The assumptions made in this paper regarding the rates are:

1. R1 rate is 3.125 hz (320 msec).
2. R3 rate is 12.5 hz (80 msec).
3. R4 rate is 25 hz (40 msec).

SCC, the high level fault handling program runs at R1 rate, i.e. every 320 msec this program processes the error information supplied by hardware. SCC also aids in surfacing the faults by running self testing programs and activating spare units at regular intervals. We can summarize the fault detection process as the arrival of disagreement at the voters of a triad, stimulated by normal activity or test activity. Test activity includes self

testing and spare cycling phases. The detection of faults initiates fault identification and later reconfiguration. The identification and reconfiguration is done with the help of special procedures initiated by SCC. To have an accurate prediction of fault detection times, for various faults it is important to know how the tasks constituting normal activity and test activity are dispatched. The flow chart in Figure 1 explains SCCs dispatching strategy at a high level. We see that test activity is dependent on a parameter set in software (Time to Cycle). Whenever a swap command is executed this parameter is initialised. In the current configuration Time to Cycle will be true every 5 seconds. When this boolean value is true, spare cycling is done. The purpose of test activity is to propagate the affects of any faults in spare units to the error latches so that SCC detects them.

DESCRIPTION OF SELF TESTING IN FTMP

Self Test programs are run to detect some of the latent faults in error latches, voters, error decoding circuitry, and cache PROMs of the FTMP. SCC calls the master self test program (SELF-TEST) which in turn calls one of the 38 self tests. Each self test is designed to test a specific unit and in each run of SCC only one of the tests is invoked. The P, R, and T tests are invoked if three corresponding bus lines are active. The C test is performed if 4 clock lines are active. Essentially in P, R, T, and C tests a disagreeing input stream is fed on one of the active lines and error latch contents are checked to see whether the injected fault is reported. In PROM test a checksum verification is made on different segments. To simplify the analysis some assumptions were made regarding the self tests.

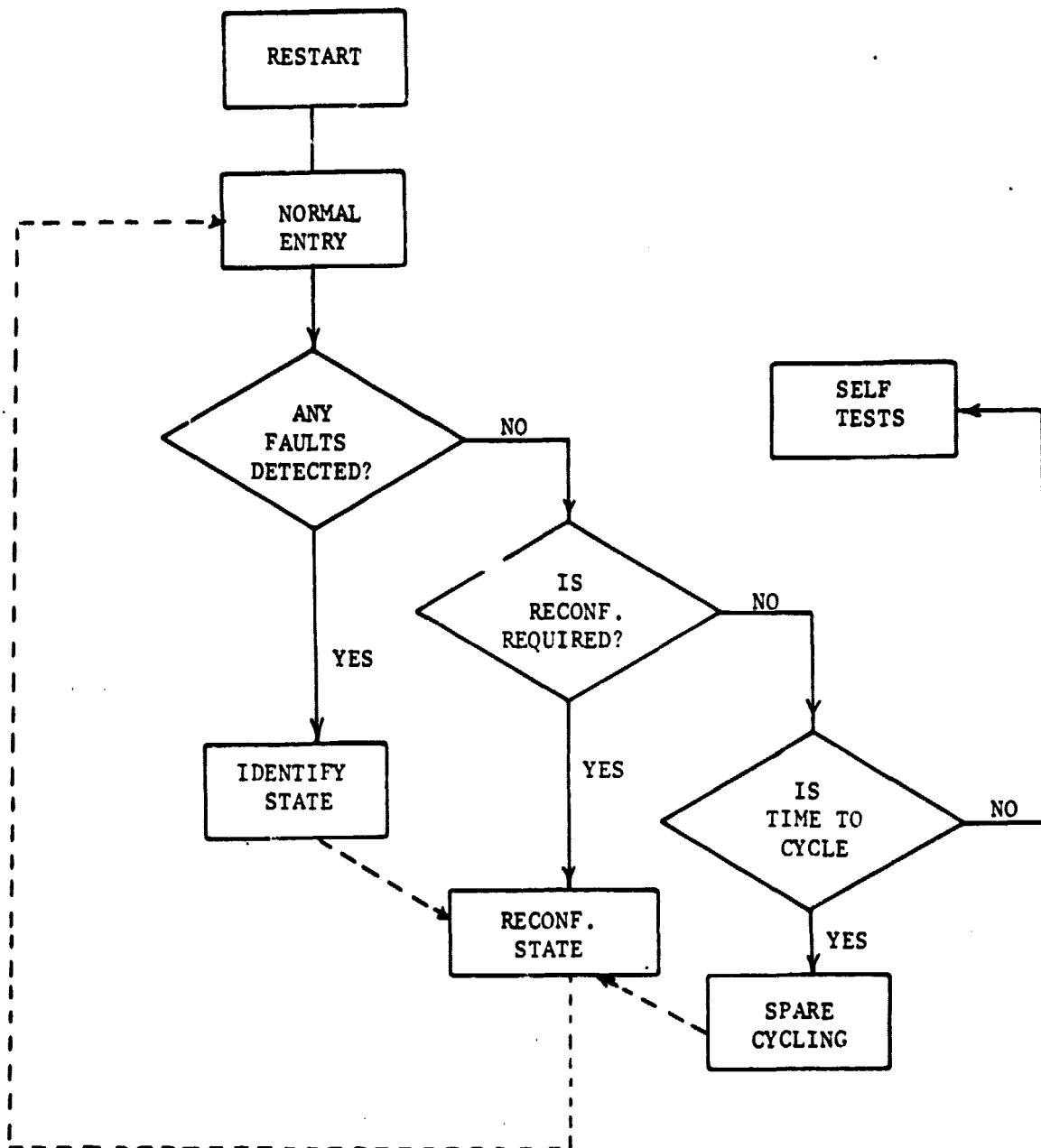


Figure 1.

ASSUMPTIONS ON SELF TESTS

As can be seen from flow chart in Figure 2 in some cycles SCC does not invoke self tests. Since the rate of self testing is much larger than spare cycling, it is assumed that self tests run periodically without interruption from Spare Cycling. This will give estimates which are lower bounds on average times. Based on this, the assumptions ST1 and ST2 are made.

ST1. Self tests are run at R1 rate, i.e., every 320 msecs.

ST2. Time to complete one cycle of self tests = $38 * 320 \text{ msecs} = 12.160 \text{ secs}$. The order is given in Figures 2 and 3.

Occurrence of a fault is random with respect to the self test cycle. The segmented testing employed makes the detection time vary between zero (when occurrence of detectable fault is immediately followed by its detecting self test) and the time taken to complete the whole cycle (when the detecting self test runs just prior to the occurrence of its detectable fault). ST3 follows from the above.

ST3. Mean time to detection of any fault which can be detected in one self test cycle = $38 * 320/2 = 6.08 \text{ secs}$.

DESCRIPTION OF SPARE CYCLING IN FTMP

FTMP has processors, memories, and buses in its spare pool. Units from the pool can be brought on line to replace any failed active unit. To uncover latent faults spares are periodically brought online even if all active units are functioning correctly. Spares are assigned as shadows to active triads. A shadow essentially duplicates the activities of the triad it is assigned to track and differs from active units in its access priorities to the buslines (e.g., a processor shadow cannot participate in polling, thus denying it access to the transmit bus). But shadows watch the R and C bus lines to

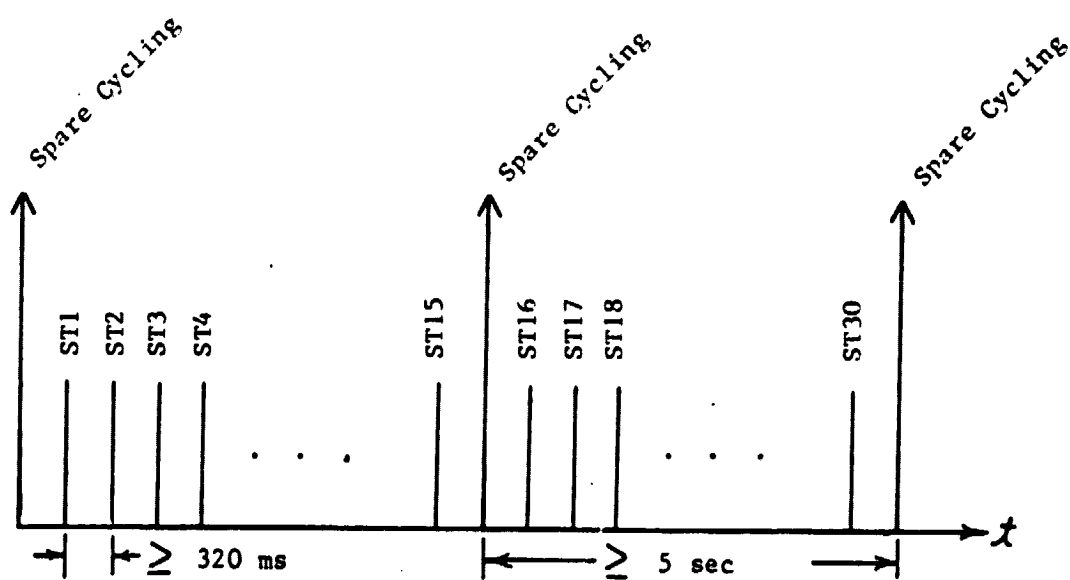


Figure 2.

Schedule of Self Tests

1. Reset self test state.
2. POLL, ANY TRIAD, LOW ORDER
3. PROM, TRIAD1, 1800
4. PROM, TRIAD2, 1800
5. PROM, TRIAD3, 1800
6. R, TRIAD1, LOW ORDER
7. R, TRIAD2, LOW ORDER
8. R, TRIAD3, LOW ORDER
9. PROM, TRIAD1, 1A00
10. T, TRIAD1, LOW ORDER
11. T, TRIAD2, LOW ORDER
12. T, TRIAD3, LOW ORDER
13. C, ANY TRIAD, LOWEST ORDER
14. POLL, TRIAD1, MIDDLE ORDER
15. PROM, TRIAD2, 1A00
16. PROM, TRIAD3, 1A00
17. PROM, TRIAD1, 1C00
18. R, TRIAD1, MIDDLE ORDER
19. R, TRIAD2, MIDDLE ORDER
20. R, TRIAD3, MIDDLE ORDER
21. PROM, TRIAD2, 1C00
22. T, TRIAD1, MIDDLE ORDER
23. T, TRIAD2, MIDDLE ORDER
24. T, TRIAD3, MIDDLE ORDER
25. C, ANY TRIAD, 2ND LOW
26. POLL, TRIAD1, HIGH ORDER
27. PROM, TRIAD3, 1C00
28. PROM, TRIAD1, 1E00
29. PROM, TRIAD2, 1E00
30. R, TRIAD1, HIGH ORDER
31. R, TRIAD2, HIGH ORDER
32. R, TRIAD3, HIGH ORDER
33. PROM, TRIAD3, 1E00
34. T, TRIAD1, HIGH ORDER
35. T, TRIAD2, HIGH ORDER
36. T, TRIAD3, HIGH ORDER
37. C, ANY TRIAD, 2ND HIGH
38. C, ANY TRIAD, HIGH ORDER

Figure 3. Schedule of self test program units

maintain synchronism. To get them on-line, only the BGUs of the shadow and the unit it is replacing have to be written into. SCC calls a procedure ISSUE-SWAP-CMND to perform spare cycling. This is issued whenever Time to Cycle is true (see flow chart). This procedure determines which spare unit should be brought on-line and which unit it should replace. On each pass only one spare unit is swapped. The order in which spare cycling is done is given in Figure 4.

ASSUMPTIONS ON SPARE CYCLING

In the present system "Time to Cycle" is true every 5 secs in the absence of reported faults. There is an interaction between spare cycling and self testing because both of them cannot be done in a single cycle. Spare cycling occurs at least twice in a cycle of self tests. Since the spare cycling rate is much smaller in comparison to self testing, we assume both self testing and spare cycling occur periodically without any interaction. This will result in a more optimistic estimate or a lower bound on the latency. Moreover the faults which can be propagated by spare cycling are configuration dependent. Cycling changes inactive units, of a particular configuration into active by changing the system configuration. This makes the otherwise latent faults get detected in subsequent normal activity. The swapping of processors and memories depends on the replacement policy. Swapping of bus lines is easier to visualise as the maximum number of spare bus lines of a particular type can be two. The following assumptions are made regarding the rate and mode of spare cycling.

SC1. Spare cycling is done every 5 secs.

SC2. Spare cycling follows the order shown in Figure 4.

SC3. After the decision on which triad to issue swap command is made,

ORIGINAL PAGE IS
OF POOR QUALITY.

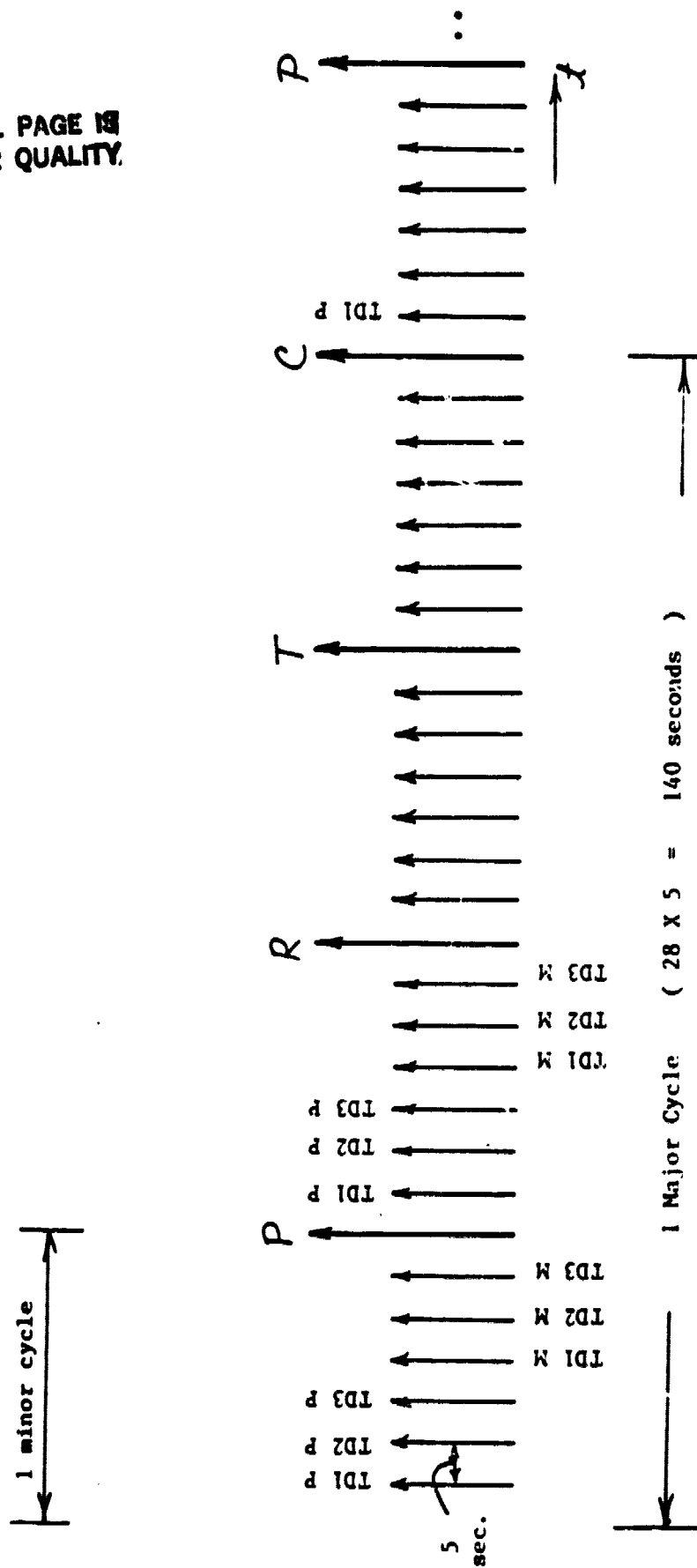


Figure 4

the units are swapped using the LRU (Least Recently Used) algorithm. Here the unit which will be used to replace is the one which has been longest in spare pool for that triad and the unit it will replace is the one which is active for the longest time.

Figures 5a and 5b illustrate this assumption as applied to processors and buses respectively. We have illustrated the case of 12 processors with no failed processors or buses.

FAULT CLASSIFICATION

It is clear that SCC detects many different types of faults and it uses various programs to uncover them. Classification of faults is based on the type of faults which would be detected by a particular detection program. Faults which go undetected are also classified. A level tree diagram (Figure 6) is given which describes the nature and difficulty in detecting a fault.

Class 1: This corresponds to those faults which become visible during the normal activity in the form of disagreement at voter propagated to the error latches.

Class 2: Faults in redundant hardware, PROMs, error latches which do not show up in normal activity belongs to this class. Self tests uncover these faults.

Class 3: Faults in inactive units serving as spares belong to this class. Spare Cycling uncovers these faults.

Class 4: Faults which go undetected.

EXAMPLE TO DEMONSTRATE FUNCTIONAL FAULT MODELING

In this section we choose a specific unit for functional fault modeling. The procedure similar to this can be adopted to compute the fault detection times for any other unit. A broad classification of the functional misbehavior patterns of the unit in question is done. The classification should be realistic in the sense that physical or logical gate level, faults could produce that kind of functional misbehavior. After having defined the misbehavior patterns one can categorize them into the fault classes discussed in the previous sections depending upon how and when they set the system error latches. To obtain the distribution of various faults amongst the fault classes one has to identify the number of faults which result in a particular misbehavior pattern. Then by exhausting all possible misbehavior patterns one can obtain the resulting distribution faults belonging to each class. The resulting distribution faults coupled with the assumptions of fault detection software made in earlier sections can be used to obtain curves for fault detection times.

We choose the 3/5 input-select unit given in Figure 7 which provides a representative partitioning of the fault classes. This unit is present in all the bus interfaces and BCU's. Its function is to select 3 out of 5 input lines based on the 4 bit select code supplied by the control register (see Table 1). Regardless of the exact nature of implementation, the following functional fault modeling approach can be taken.

- MB1. Selects less than three active lines (assuming no duplication of any active line).
- MB2. Duplicated active line comprises two of the three active lines selected.

MB3. Faults which result in functional misbehavior only when the select code or system configuration is changed (faults here manifest as MB1 or MB2 when configuration is changed).

MB4. Faults which cannot be detected.

Gate level faults can be mapped into the above functional fault classification.

Except for the BCU's, all other 3/5 select units are used in conjunction with a voter-error ROM combination and MB1 lines up with Class 1. These functional classes MB1 will fall into the earlier overall fault classes depending upon the location of the 3/5 select unit. If the select unit is located in a spare then MB1, MB2, and MB3 are in Class 3, faults in inactive units. When the select unit is in a BCU, MB1 falls into Class 2 which requires special testing.

To illustrate the many to one mapping we take a specific implementation of the 3/5 select (T BUS INTERFACE) and map the gate level faults to MB1, MB2, MB3, and MB4 (Class1, Class2, Class3 and Class4 respectively). Some faults are data dependent and map to different fault behavior for different implementations. To illustrate this, let the initial select code correspond to lines 1, 2, and 4 being active. Owing to some fault if 1, 3, and 3 are selected then clearly it falls to MB1 in say the T bus interface. But if the initial configuration changes from 1, 2, and 3 to 1, 3, and 3 due to a fault then the fault falls to MB2. The duplication of a line for a select code is implementation dependent. Therefore it is better to consider implementation details to obtain a finer model. Assuming that all possible errors can occur in the select code we tabulate the faults. Since the select code at a particular instant of time is not known an averaging technique is used. We assume that all valid codes are equally likely. By this we get distribution of select code faults into various classes (see Table 2).

Select Code				For U_{29}		U_{30}					
S3	S2	S1	S0	A	B	A	B	Line 1	Line 2	Line 3	
				(S0) (S ₁ S ₂)		(S1) (S2)					
0	0	0	0	0	1	0	0	4	2	1	*
0	0	0	1	1	1	0	0	5	2	1	*
0	0	1	0	0	1	0	1	4	3	1	*
0	0	1	1	1	1	0	1	5	3	1	*
0	1	0	0	0	1	1	0	4	3	2	*
0	1	0	1	1	1	1	0	5	3	2	*
0	1	1	0	0	0	1	1	2	5	4	*
0	1	1	1	1	0	1	1	3	5	4	*
1	0	0	0	0	1	0	0	3	2	1	*
1	0	0	1	1	1	0	0	3	2	1	
1	0	1	0	0	1	0	1	3	3	1	
1	0	1	1	1	1	0	1	3	3	1	
1	1	0	0	0	1	1	0	3	3	2	
1	1	0	1	1	1	1	0	3	3	2	
1	1	1	0	0	0	1	1	1	5	4	
1	1	1	1	1	0	1	1	1	5	4	*

* Valid Codes

Table 1

INITIAL CODE	# CLASS 1	#CLASS 2	# CLASS 3	# CLASS 4
0000	15	0	0	0
0001	15	0	0	0
0010	13	2	0	0
0011	13	2	0	0
0100	13	2	0	0
0101	13	2	0	0
0110	15	0	0	0
0111	15	0	0	0
1000	10	4	1	0
1111	14	0	1	0
<hr/>				
Average	13.6	1.2	0.2	0
<hr/>				

Table 2. Select code input error behavior

In a similar vein we can get distribution of input and output line stuck faults. Here we assume that the select code is not faulty and faults manifest on input and output lines. Faults assumed for input and output signal lines are stuck-at-1, stuck-at-0 or inversion of the signal line. The number of fault cases for each legal code is given in Table 3.

SELECT CODE	# CLASS 1	# CLASS 2	#CLASS 3	# CLASS 4
0000	18	0	33	6
0001	18	0	33	6
0010	18	0	33	6
0011	18	0	33	6
0100	18	0	33	6
0101	18	0	33	6
0110	18	0	33	6
0111	18	0	33	6
1000	18	0	33	6
1111	18	0	33	6
<hr/>				
Average	18	0	33	6
<hr/>				

Table 3. Input or output faults

Combining tables 2 and 3 we get the following distribution of faults:

Class 1	43.88%
Class 2	1.66%
Class 3	46.11%
Class 4	8.33%

Table 4.

Class 1 faults will be detected between 0-320 msec. Class 2 faults here would require error-latch and voters self tests. Based on the assumptions made earlier, the Class 2 faults will be detected between 0-12.16 sec.

Class 3 fault detection in the case of 3/5 select unit requires cycling spare lines. From Figure 5b time between two bus swap commands is 140 secs. This means that half of the Class 3 errors could be detected in a maximum time of 140 secs and the rest after the next 140 secs. This case is plotted in Figure 8.

We summarize the assumptions for Figure 8:

1. The faults are equally likely input select code errors and equally likely bus input and select output stuck errors.
2. The select unit is not in a BGV but is equally likely to be any other location.
3. Spare cycling and self-test run at their most frequent rate, i.e., the system is lightly loaded.

	Triad 1	Triad 2	Triad 3
A	1 2 3 (10)	4 5 6 (11)	7 8 9 (12)
B	10 2 3 (1)*	4 5 6 (11)	7 8 9 (12)
C	10 2 3 (1)	11 5 6 (4)*	7 8 9 (12)
D	10 2 3 (1)	11 5 6 (4)	12 8 9 (7)*
E	10 1 3 (2)*	11 5 6 (4)	12 8 9 (7)
F	10 1 3 (2)	11 4 6 (5)*	12 8 9 (7)
G	10 1 3 (2)	11 4 6 (5)	12 7 9 (8)*
H	10 1 2 (3)*	11 4 6 (5)	12 7 9 (8)
I	10 1 2 (3)	11 4 5 (6)*	12 7 9 (8)
J	10 1 2 (3)	11 4 5 (6)	12 7 8 (9)*
K	1 2 3 (10)*	11 4 5 (6)	12 7 8 (9)
L	1 2 3 (10)	4 5 6 (11)*	12 7 8 (9)
M	1 2 3 (10)	4 5 6 (11)	7 8 9 (12)*

* Swap of shadow and active unit

Figure 5(a) Processor spare cycling assuming 12 processors.

One major cycle.

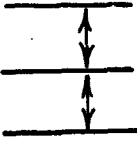
Bus Sparer	Active Bus Lines	
(5 4)	3 2 1	
(1 5)	4 3 2	
(2 1)	5 4 3	
(3 2)	1 5 4	
(4 3)	2 1 5	
(5 4)	3 2 1	

Figure 5(b)

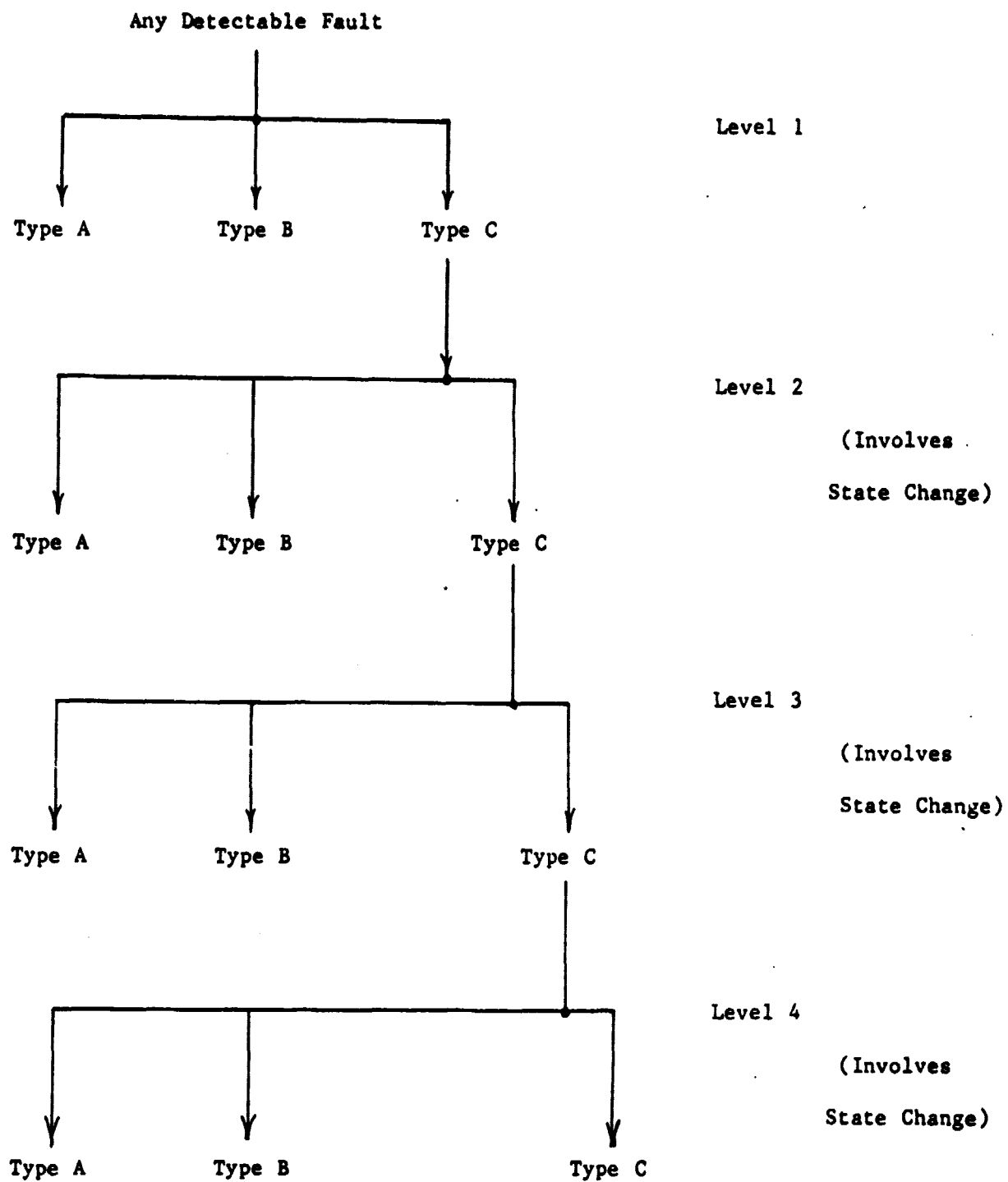


Figure 6.
(Level Tree Diagram)

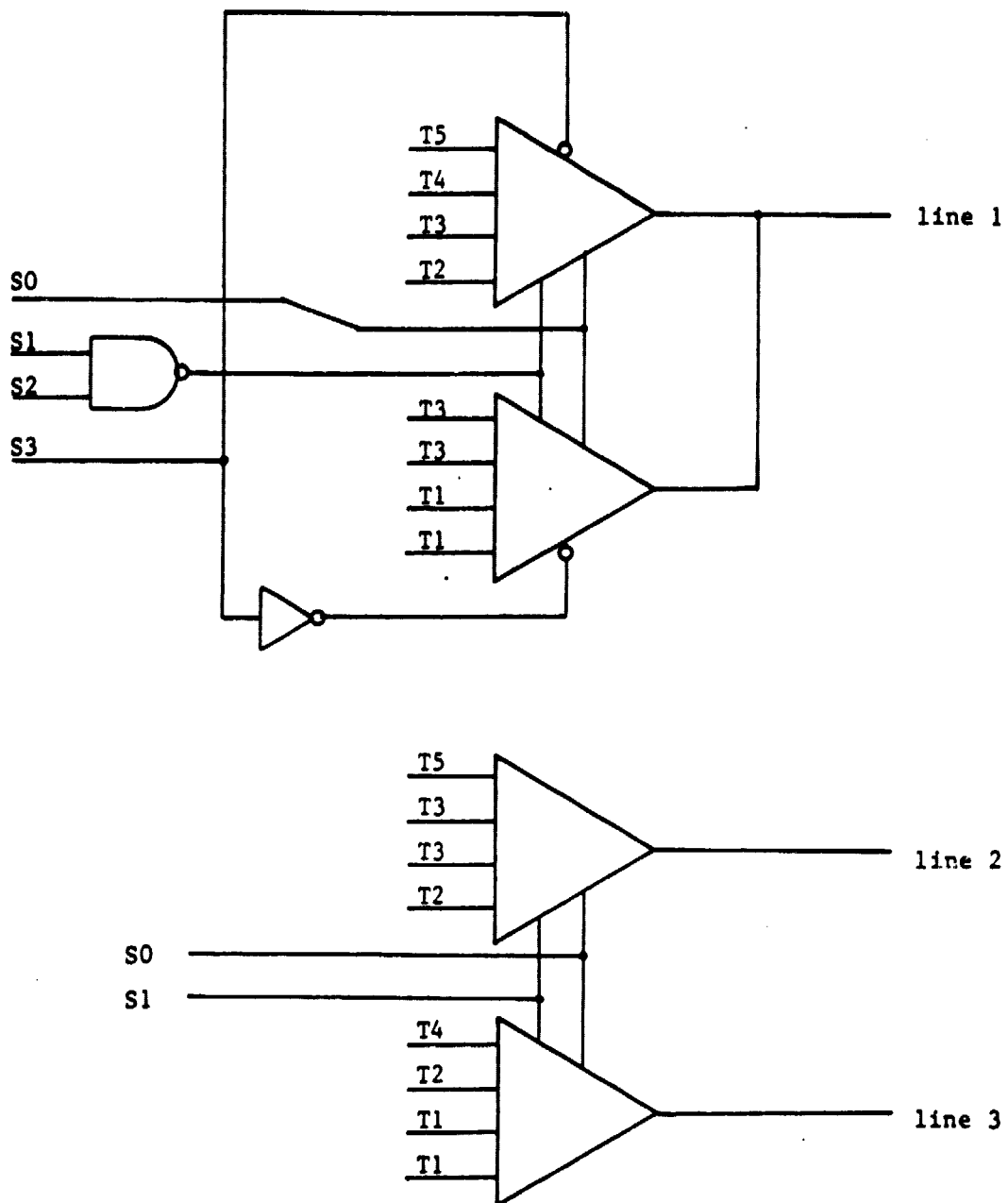
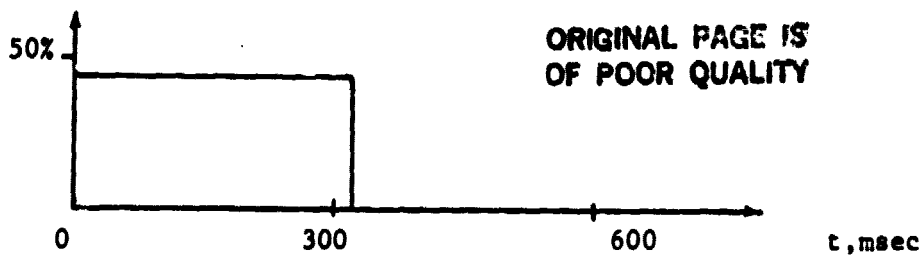
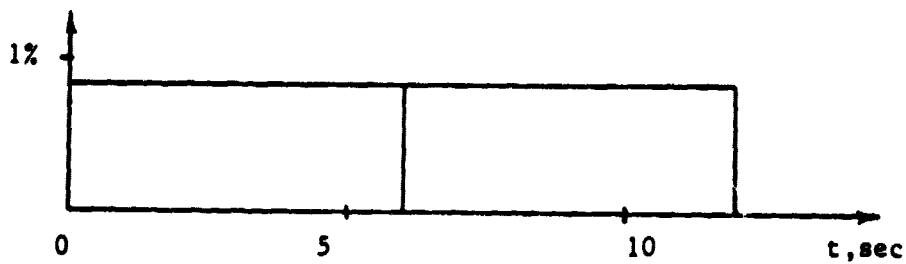


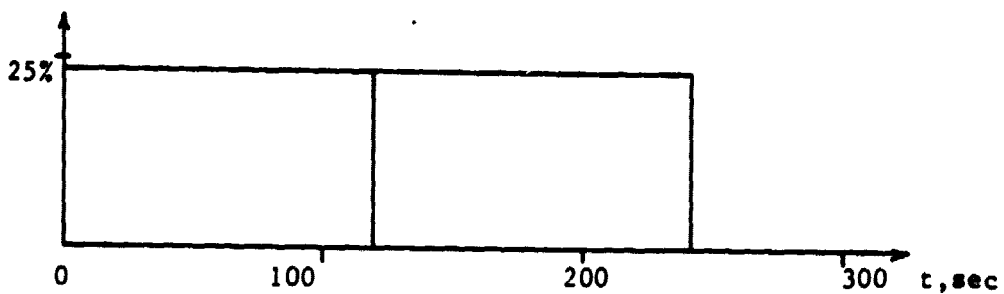
Figure 7. Three out of five select unit (T Bus interface)



a. Class 1



b. Class 2



c. Class 3



d. Total of all classes

Figure 8 Detection time for 3/5 select

LATENCY

We next relate the functional fault classes and latency estimates of the 3/5 select unit to a system model. Faults have significantly different effects on system behavior. For example a long latency fault in a nonactive processor doesn't influence the system until it is reconfigured as an active unit; but the failure of an active P bus line has immediate consequences. Hence the functional fault classes ought to be of sufficient detail to allow appropriate assignment to the various transitions in the system reliability mode (e.g. Care III).

The classification used in arriving at Tables 2, 3, and 4 would be applicable to an FTMP system reliability model which distinguished between faults in active processors and memories and spare processors and memories. But the classification is not of sufficient detail if BGU behavior is explicitly included in the reliability model. Specifically the functional behavior MB1 (selecting less than 3 active lines) should be subdivided for the BGU into the selection of 2 active lines and less than two active lines. The BGU behavior differs for these two subcases.

Suppose the reliability model does distinguish between active processors and spare processors, then the Classes in Figure 8 would probably be assigned as follows:

1. Class 1 and Class 2 to transitions for a fault in an active processor.
2. Class 3 to transitions for a fault in a spare processor.
3. Class 4 to reduce the occurrence probability of a fault.

MODELING PROCESS

The previous sections have attempted to illustrate the functional fault modeling process when applied to a particular unit the estimation of fault latency. The following factors are important in this modeling.

1. The detail known about the physical devices and structure of the implementation and the possible fault mechanisms.
2. The hardware and software structure used to detect that a fault has occurred. What redundancy exists in space and time and how is it used to detect faults.
3. The proposed use to be made of the latency estimates and the level of detail about fault behavior that is required. For example, the determination of worst case behavior.

REFERENCES

1. A.L. Hopkins, Jr., T.B. Smith, and J.H. Lala, "FTMP — A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", Proceedings of the IEEE, October 1978, pp. 1221-1239.
2. S.J. Bavuso, "Advanced Reliability Modeling of Fault-Tolerant Computer-Based Systems", NATO Advanced Study Institute, Norwich UK, July 1982.
3. P.R. Menon and S.G. Chappell, "Deductive Fault Simulation with Functional Blocks", IEEE Trans. Computers, vol. C-27, pp. 689-695, August 1978.
4. M.A. Breuer and A.D. Friedman, "Functional Level Primitives in Test Generation", IEEE Trans. Computers, pp. 223-235, March 1980.
5. A. Miczo, "Fault Modelling for Functional Primitives", 1982 IEEE Test Conference, pp. 43-49, 1982.
6. S.M. Reddy, "Complete Test Sets for Logic Functions", IEEE Trans. Computers, Vol C-22, pp. 1016-1020, Nov. 1973.
7. S.B. Akers, "Universal Test Sets for Logic Networks", IEEE Trans. Computers, Vol. C-22, pp. 835-839, September 1973.
8. E. Konemann et al., "Built-in Test for Complex Digital Integrated Circuits", IEEE J. of Solid State Circuits, pp. 315-319, June 1980.
9. Y.H. Levendel and P.R. Menon, "Test Generation Algorithms for Computer Hardware Description Languages", IEEE Trans. Computers, pp. 577-588, July 1982.
10. W.C. Carter, "Signature Testing with Guaranteed Bounds for Fault Coverage", 1982 IEEE Test Conference, pp. 75-82, 1982.
11. K. Son and J.Y.O. Fong, "Automatic Behavioral Test Generation", 1982 IEEE Test Conference, pp. 161-165, 1982.